SOFTWARE ENGINEERING IN TEST

Pre-screening: Assignment

Matthew Thill

TABLE OF CONTENTS

PROBLEM SET #1 ANAGRAM TEST 1

Requirement: 1
Planning 2
Code 3
Testing 4
Test Results 4

PROBLEM SET #2 REPEATING ELEMENT IN ARRAY 6

Requirement: 6
Planning 7
Code 8
Testing 9
Test Results 9

PROBLEM SET #3 FIND TRIPLET 11

Requirement: 11
Planning 12
Code 13
Testing 14
Test Results 14

PROBLEM SET #1
ANAGRAM TEST

REQUIREMENT:

Write a function to check whether two given strings are anagram of each other or not

What is anagram?

An anagram of a string is another string that contains same characters, only the order of characters can be different.

Example:

I/P: Str1= “abcd”, Str2= “dabc”

O/P: Input strings are anagram of each other

PLANNING

We need to be able to compare two strings without worrying about uppercase, lower case, and any white spaces.

- *Need to strip white spaces, leading and trailing spaces*
- *Need to convert to lower case for both strings for easy comparison.*
- *Most importantly, we need to re-order the characters (e.g. alphabetically)*
    - *e.g. 'dbca' → 'abcd'*
- *We can then compare if both strings are the same if:*
    - *All the characters match up*
    - *The total number of characters in the strings must be the same*
    - *We also need to set the minimum supported anagram string to two chars.*
        - *e.g. 'he' → 'eh'*
        - *After all, one char doesn't make sense.*
            - *Display an appropriate message for not supporting this.*
    - *Display proper message If both strings specified are similar:*
        - *e.g. string_1 = ' a b c d ' and string_2 = 'abcd'*
            - *Notice the white spaces and leading and trailing spaces?*
    - *It's possible to support special characters in the string, so don't limit the strings to only alphabet letters.*
        - *e.g. "friends'", or "Friend's"*

CODE

Please see the python script "./problem_1/***anagram_validation.py***" that was submitted along with this report.

```
#These are the output messages we want to display, so let's store them in seperate meaningful variables - useful for unittest as well.
global msg_min_char_required
msg_min_char_required = "String supplied must contain at least 2 letters. Please check your strings and try again."
global msg_identical_char_order
msg_identical_char_order = "The supplied strings are close to identical. Please supply a string with different order of character(s)."
global msg_match
msg_match = "Input strings are anagram of each other"
global msg_no_match
msg_no_match = "Input strings are NOT anagram of each other"


#Function: anagram_validator
#INPUT: Takes in two strings - e.g. Str1= "abcd", Str2= "dabc"
#OUTPUT: Returns a string - a msg as defined in the above global variables.
def anagram_validator(string1, string2):
    print("\n\nCOMPARING: \n\t>1st String: \"{}\" \n\t>2nd String: \"{}\"".format(string1,string2))

    #Anagrams strings can contain spaces, and different case chars, but for easy comparison let's remove the spaces and make them all lowercase for both strings.
    str1 = convert_to_no_space_lower_string(string1)
    str2 = convert_to_no_space_lower_string(string2)

    # validate the strings supplied are infact alphabets only.
    # Validate for alphabets only (NOTE: it's possible for anagrams to contain non-alhpabets - e.g. friends' or friend's)
    # if (convertToLowerAlpha(.isalpha()) or (Str2.isalpha()):
    #  outputString = "Non-Alphabets detected in supplied strings. Please check your strings and try again."

    msg = msg_no_match                              # Default msg when the strings are not anagrams.

    if ((len(str1) < 2) or (len(str2) < 2)):        # Lowest possible anagram string should contain at minimum 2 chars.
        msg = msg_min_char_required
    elif (len(str1) == len(str2)):                  # For it to be an anagram string, both strings character count should match, not including white spaces.
        if (str1 == str2):                          # If both strings have same char order, they can't be an anagram, they are too close to being identical
            msg = msg_identical_char_order
        elif (is_same_sorted_string(str1, str2)):   # Sort the order of the chars (without the spaces, all lowercase) to see if both strings have the same char order
            msg = msg_match
    print ("\t=>RESPONSE: "+msg)
    return msg


#Function: convert_to_no_space_lower_string
#INPUT: Takes in a strings - e.g. Str1= "abcd" or Str2= "dabc"
#OUTPUT: Returns a string with no white spaces, and in lowercase.
def convert_to_no_space_lower_string(str):
    # remove leading & trailing spaces
    str = str.strip()
    # remove blank spaces
    str = str.replace(" ", "")
    # return a lowercase string for easy comparison.
    return str.lower()


#Function: is_same_sorted_string
#INPUT: Takes in 2 strings - e.g. Str1= "abcd" or Str2= "dabc"
#OUTPUT: Returns a boolean - True if both list (convered from string) match and False if they don't.
def is_same_sorted_string(str1, str2):
    # compare two lists, are they the same? e.g. ['a', 'b', 'c', 'd'] == ['w', x', 'y', 'z'] ?
    return (sort_string(str1) == sort_string(str2))


#Function: sort_string
#INPUT: Takes in a strings - e.g. "abcd" or "dabc"
#OUTPUT: Returns a sorted list - e.g "dabc" -to-> ['a', 'b', 'c', 'd']
def sort_string(str):
    # Take the iterable string 'myString', and return a new sorted list
    return sorted(str)

```

TESTING

The test table below covers a wide range of test scenarios – includes both positive and negative tests.

TEST RESULTS

The following test results were extracted from the unittest created for this problem set. Please see the python script "./problem_1/***test_anagram_validation.py***" that was submitted along with this report.

Test Case #	String 1	String 2	Expected (Anagram?)	Actual (Anagram?)	PASS / FAIL
1	abcd	dabc	YES	YES	PASS
2	listen	silent	YES	YES	PASS
3	hydroxydeoxycorticosterones	hydroxydesoxycorticosterone	YES	YES	PASS
4	Tom Marvolo Riddle	I am Lord Voldemort	YES	YES	PASS
5	rail safety	fairy tales	YES	YES	PASS
6	friend's	friends'	YES	YES	PASS
7	ab	ba	YES	YES	PASS
8	cow	cows	NO	NO	PASS
9	radio	frequency	NO	NO	PASS
10	planet	hearts	NO	NO	PASS
11	Lorem Ipsum is simply dummy text	printing and typesetting industry	NO	NO	PASS
12	<_)*$%^&#@!~	abc	NO	NO	PASS
13	ee	ee	NO	NO	PASS
14	us	us	NO	NO	PASS
15	<empty> - ''	<empty> - ''	NO	NO	PASS
16	<empty> - ''	dabc	NO	NO	PASS
17	abcd	<empty> - ''	NO	NO	PASS
18	a	d	NO	NO	PASS
19	e	e	NO	NO	PASS

Sample: The following is a snippet of the unittest result from "./problem_1/***test_anagram_validation.py***".

(Please note: there are more unittests for this problem set than what the snippet shows)

```
COMPARING:
        >1st String: "e"
        >2nd String: "e"
        =>RESPONSE: String supplied must contain at least 2 letters. Please check your strings and try again.
.
COMPARING:
        >1st String: "abcd"
        >2nd String: "dabc"
        =>RESPONSE: Input strings are anagram of each other


COMPARING:
        >1st String: "listen"
        >2nd String: "silent"
        =>RESPONSE: Input strings are anagram of each other


COMPARING:
        >1st String: "hydroxydeoxycorticosterones"
        >2nd String: "hydroxydesoxycorticosterone"
        =>RESPONSE: Input strings are anagram of each other
.
COMPARING:
        >1st String: "Tom Marvolo Riddle"
        >2nd String: "I am Lord Voldemort"
        =>RESPONSE: Input strings are anagram of each other
.
COMPARING:
        >1st String: "rail safety"
        >2nd String: "fairy tales"
        =>RESPONSE: Input strings are anagram of each other
.
COMPARING:
        >1st String: "friend's"
        >2nd String: "friends'"
        =>RESPONSE: Input strings are anagram of each other
.
COMPARING:
        >1st String: "ab"
        >2nd String: "ba"
        =>RESPONSE: Input strings are anagram of each other
.
----------------------------------------------------------------------
Ran 8 tests in 0.002s

OK
```

PROBLEM SET #2
REPEATING ELEMENT IN ARRAY

REQUIREMENT:

Find the first repeating element in an array of integers

Given an array of integers, find the first repeating element in it. We need to find the element that occurs more than once and whose index of first occurrence is smallest.

Example:

I/P: arr=[100,5,3,1,0,5,-8]

O/P: 5 [5 is the first element that repeats]

PLANNING

For a given array arr=[100,5,3,1,0,5,-8], we need to be able to compare every element in the array

- *We need a reference to compare against the rest of the elements in the array.*
- *Which means we need to loop through the entire array comparing the reference with the next set of elements.*
  - *Comparing from left to right.*
    - *Left to right because we want the smallest index of the first repeating element.*
  - *Reference would have to change from the first index (e.g. n) to the 2nd last index in the array (e.g. n-1).*
  - *We would need two counters: one to compare with the other*
    - *One would be the reference counter*
    - *The other would be the current counter.*
  - *We also need a way to flag once the first element has been found.*
  - *If no repeating element has been found:*
    - *The initial flag value to be set to a not found*
    - *a proper message should be displayed – .e.g. no repeating element found.*

CODE

Please see the python script "./problem_2/***first_repeating_element.py***" that was submitted along with this report.

```python
#Function: find_first_repeating_number
#INPUT: Takes in an array of integers. e.g. [100,5,3,1,0,5,-8]
#OUTPUT: Returns a list '[boolean, the message, the repeating #):
#   find_first_repeating_number(arr)[0]: Boolean - True or False if a repeating number found.
#   find_first_repeating_number(arr)[1]: Exact message displayed when found or not.
#   find_first_repeating_number(arr)[2]: Repeating # found; if no repeat #, use empty str (''); if find_first_repeating_number(arr)[0] is True, return #.
def find_first_repeating_number(arr):
    print("\n\nARRAY SOURCE: \t{}".format(arr))

    #Flag initially set to not found - e.g. False.
    is_repeating = False

    #Outer loop index, "ref_index" (or ref. counter) used as a point of reference to compare against inner loop.
    #We have an outer loop and an inner loop index, and they should never overlap. So when we reach the end of an
    #array (e.g. the last element in the array) the outer loop would never point to the last element as it will
    #always be one less - e.g. outer loop index/counter would be 'n-1' (when we get to the last cycle in the loops)
    # when inner loop index/counter is 'n'.
    for ref_index in range(0, len(arr) - 1):
#         print("\n*** DEBUGGING: ref_index: "+str(arr[ref_index]))

        #Inner loop: traverses through the array to verify if there is a matching number w/ the outer loop index
        #Inner loop should shrink from the left as the offset of ref_index moves one index over to the right.
        for current_index in range (ref_index + 1, len(arr)):
#             print("\t*** DEBUGGING: current_index: "+str(arr[current_index])+"\n\t\t\tremaining to verify in inner loop: "+(str(arr[current_index:])))

            #MATCH: When the reference index value and the current index value in the inner loop are the same.
            if (arr[ref_index] == arr[current_index]):
#                 print("\n\t\t*** DEBUGGING: REPEATING NUMBER FOUND!!! ***")
                msg = "\t=>RESULT: {} is the first element that repeats".format(arr[current_index])
                print (msg)
                is_repeating = True
                #no need to traverse through the loops as the first repeating element has been found!
                return [is_repeating, msg, arr[current_index]]

    if not is_repeating:    #If no repeating integers found display not found msg.
        msg = "\t=>RESULT: No repeating element was found in the array."
        print (msg)

    #For consistancey we need a 3rd element [2], even if we find no repeating number, we must be aware '' is just a placeholder and not
    #actually the repeating #. Especially since element [0] has been flagged as False - no repeat found in the array.
    return [is_repeating, msg, '']
```

TESTING

The test table below covers a wide range of test scenarios – includes both positive and negative tests.

TEST RESULTS

The following test results were extracted from the unittest created for this problem set. Please see to the python script "./problem_2/***test_first_repeating_element.py***" that was submitted along with this report.

Test Case #	Array Source	Expected (Repeating?)	Actual (Repeating?)	PASS / FAIL
1	[100, 5, 3, 1, 0, 5, -8]	YES	YES	PASS
2	[100, -8, 3, 1, 0, -5, -8, 9, 4, 3, 2, 6, 4, 5, 0, -5, 2, -1, 0, 6, 7, 1, -200]	YES	YES	PASS
3	[100, 3, 3, 1, 0, 5, -8, 9, 4, 3, 2, 6, 4, 5, 0, -5, 2, -1, 0, 6, 7, 1, -200]	YES	YES	PASS
4	[100, 100, 3, 1, 0, 5, -8, 9, 4, 3, 2, 6, 4, 5, 0, -5, 2, -1, 0, 6, 7, 1, -200]	YES	YES	PASS
5	[100, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 100]	YES	YES	PASS
6	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 100, 100]	YES	YES	PASS
7	[0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 0, 1, 0]	YES	YES	PASS
8	[10, 10]	YES	YES	PASS
9	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 100]	NO	NO	PASS
10	[]	NO	NO	PASS
11	[100]	NO	NO	PASS

Sample: The following is a snippet of the unittest result from "./problem_2/***test_first_repeating_element.py***".

(Please note: there may be more unittests for this problem set than what the snippet shows)

```
ARRAY SOURCE:   [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 100]
        =>RESULT: No repeating element was found in the array.
.
ARRAY SOURCE:   []
        =>RESULT: No repeating element was found in the array.

ARRAY SOURCE:   [100]
        =>RESULT: No repeating element was found in the array.
.
ARRAY SOURCE:   [100, 5, 3, 1, 0, 5, -8]
        =>RESULT: 5 is the first element that repeats

ARRAY SOURCE:   [100, -8, 3, 1, 0, -5, -8, 9, 4, 3, 2, 6, 4, 5, 0, -5, 2, -1, 0, 6, 7, 1, -200]
        =>RESULT: -8 is the first element that repeats

ARRAY SOURCE:   [100, 3, 3, 1, 0, 5, -8, 9, 4, 3, 2, 6, 4, 5, 0, -5, 2, -1, 0, 6, 7, 1, -200]
        =>RESULT: 3 is the first element that repeats

ARRAY SOURCE:   [100, 100, 3, 1, 0, 5, -8, 9, 4, 3, 2, 6, 4, 5, 0, -5, 2, -1, 0, 6, 7, 1, -200]
        =>RESULT: 100 is the first element that repeats

ARRAY SOURCE:   [100, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 100]
        =>RESULT: 100 is the first element that repeats

ARRAY SOURCE:   [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 100, 100]
        =>RESULT: 100 is the first element that repeats

ARRAY SOURCE:   [0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 0, 1, 0]
        =>RESULT: 0 is the first element that repeats
.
ARRAY SOURCE:   [10, 10]
        =>RESULT: 10 is the first element that repeats
.
----------------------------------------------------------------------
Ran 4 tests in 0.002s

OK
```

PROBLEM SET #3
FIND TRIPLET

REQUIREMENT:

Find a triplet that sum to a given value X.

Write a function to find if there is a triplet in integer array whose sum is equal to the given value x. If there is such a triplet present in array, then print the triplet and return true. Else return false. Just print first triplet if there are multiple triplets matching value x.

Example:

I/P : array [12, 3, 4, 1, 6, 9,-6] and given sum x =24

O/P : (12, 3 and 9)

PLANNING

For a given *array [12, 3, 4, 1, 6, 9,-6] and given sum x =24*, we need to be able to compare every element in the array.

- *Since a Triplet is made up of 3 numbers, we need three counters to traverse through the loop comparing to see if the sum of all 3 counters add to X.*
- *Since we'll be needing three loops (one nested in the other), this would allow us to check if the sum of the three counters add up to X.*
- *We need to ensure that our counters do not overlap at any given time, so each counter would be working within a subset of elements until all elements in the array are evaluated.*
    - *The subset would imply the starting position of the 1st counter would be at 0, 2nd counter position would be 1, and 3rd counter position would be at 2.*
        - *As 3rd counter (inner or current counter) reaches the end of the array, the next time it loops through, the position of all 3 counters should shift to the right by 1 position.*
    - *Similarly, Ending positions for counter one (outer loop) would be n-2 where, n is the element index.*
        - *So 2nd or mid counter loop would be n-1, while current counter (inner counter loop) would be n.*
    - *If the sum of all 3 elements adds up to X, then exit out of all loops and return what the triplet values are.*
    - *If no triplet is found, a valid message should be displayed.*
        - *The initial flag value to be set to a no triplet found*
        - *a proper message should be displayed – .e.g. no triplet found.*

CODE

Please see the python script "./problem_3/***find_triplet.py***" that was submitted along with this report.

```python
def get_triplet (arr, sum_x):
    print("\n\nFor ARRAY SOURCE: {}, SUM X=\"{}\":".format(arr,sum_x))

    #Flag initially set to not found - e.g. False.
    is_triplet = False

    #We have an outer loop and 2 inner loop indexes, and they should never overlap. So when we reach the end of an
    #array (e.g. the last element in the array) the outer loop would never point to the last element as it will
    #always be two less - e.g. outer loop index would be 'n-2', while mid loop index is 'n-1' and inner loop index is 'n'.
    #e.g. outer loop index would never point to the last two elements.
    for outer_loop_index in range(0, len(arr) - 2):
#         print("*** DEBUGGING: outer_loop_index: "+str(arr[outer_loop_index]))

        #Mid loop should shrink from the left as the offset of outer_loop_index moves one index over to the right.
        #Mid loop index should never point to the last element of an array.
        for mid_loop_index in range(outer_loop_index + 1, len(arr) - 1):
#             print("\t*** DEBUGGING: mid_loop_index: "+str(arr[mid_loop_index]))

            #Inner loop: traverse through the array to verify if the sum of three indexes (e.g. outer, mid & inner) make up a triplet
            #for inner_loop_index in range (outer_loop_index + 2, len(arr)):
            for inner_loop_index in range (mid_loop_index + 1, len(arr)):
#                 print("\t\t*** DEBUGGING: inner_loop_index: {} remaining to verify in inner loop: {}".format(arr[inner_loop_index], arr[inner_loop_index:]))
                #Add up all 3 values from each of the 3 different positions/index.
                triplet_sum = arr[outer_loop_index] + arr[mid_loop_index] + arr[inner_loop_index]
                if (triplet_sum == sum_x):
#                     print("\t\t\t*** DEBUGGING: TRIPLET FOUND!!! ***")
                    msg = "[{},{} and {}] are TRIPLET for the given sum x={} in the array: {}".format(arr[outer_loop_index], arr[mid_loop_index], arr[inner_loop_index], sum_x, arr)
                    print ("\t\t\t=>RESULT: {}".format(msg))

                    is_triplet = True
                    #no need to traverse through the loops if the first triplet has been found!
                    return [is_triplet, msg, [arr[outer_loop_index], arr[mid_loop_index], arr[inner_loop_index]]]


    #If no triplet found, display not found message for the corresponding array and sum, x.
    if not is_triplet:
        msg = "No triplet found for the sum x={} in the array: {}".format(sum_x, arr)
        print ("\t\t\t=>RESULT: {}".format(msg))
    #return [is_triplet, msg, [arr[outer_loop_index], arr[mid_loop_index], arr[inner_loop_index]]]

    #For consistancey we need a 3rd element [2], even if we find no triplet, we must be aware '[]' is just a place holder
    #Especially since element [0] has been flagged as False - no triplet exist in the array.
    return [is_triplet, msg, []]

```

TESTING

The test table below covers a wide range of test scenarios – includes both positive and negative tests.

TEST RESULTS

The following test results were extracted from the unittest created for this problem set. Please see the python script "./problem_3/***test_find_triplet.py***" that was submitted along with this report.

Test Case #	Array Source	Sum(X)	Actual (Triplet?)	PASS / FAIL
1	[12, 3, 4]	19	[12,3 and 4]	PASS
2	[12, 3, 4, 1, 6, 9, -6]	12	[12,6 and -6]	PASS
3	[100, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 100]	15	[1,4 and 10]	PASS
4	[100, 100, 3, 1, 0, 5, -8, 9, 4, 3, 2, 6, 4, 5, 0, -5, 2, -1, 0, 6, 7, 1, -200]	5	[3,1 and 1]	PASS
5	[100, 3, 3, 1, 0, 5, -8, 9, 4, 3, 2, 6, 4, 5, 0, -5, 2, -1, 0, 6, 7, 1, -200]	106	[100,3 and 3]	PASS
6	[100, -8, -1, -2, 0, -5, -8, -1, -4, 0, 2, -6, 4, 5, 0, -5, 2, -1, 0, -6, 7, 1, -200]	-192	[7,1 and -200]	PASS
7	[12, 3, 4, 1, 6, 9, -6]	24	[12,3 and 9]	PASS
8	[]	0	NO	PASS
9	[12]	12	NO	PASS
10	[12, 3]	15	NO	PASS
11	[12, 3, 0]	9	NO	PASS
12	[100, 5, 3, 1, 0, 5, -8]	24	NO	PASS
13	[12, 3, 4, 1, 6, 9, -6]	200	NO	PASS
14	[2, 1, 5, 1, 6, 29, -6]	6	NO	PASS
15	[2, 1, 5, 1, 6, 29, -6]	3	NO	PASS
16	[2, 1, 5, 1, 6, 29, -6]	15	NO	PASS
17	[2, 1, 5, 1, 6, 29, -6]	30	NO	PASS
18	[2, 1, 5, 1, 6, 29, -6]	87	NO	PASS
19	[2, 1, 5, 1, 6, 29, -6]	18	NO	PASS

Sample: The following is a snippet of the unittest result from "./problem_3/***test_find_triplet.py***".

(Please note: there are more unittests for this problem set than what the snippet shows)

```
For ARRAY SOURCE: [2, 1, 5, 1, 6, 29, -6], SUM X="30":
                        =>RESULT: No triplet found for the sum x=30 in the array: [2, 1, 5, 1, 6, 29, -6]


For ARRAY SOURCE: [2, 1, 5, 1, 6, 29, -6], SUM X="15":
                        =>RESULT: No triplet found for the sum x=15 in the array: [2, 1, 5, 1, 6, 29, -6]


For ARRAY SOURCE: [2, 1, 5, 1, 6, 29, -6], SUM X="3":
                        =>RESULT: No triplet found for the sum x=3 in the array: [2, 1, 5, 1, 6, 29, -6]


For ARRAY SOURCE: [2, 1, 5, 1, 6, 29, -6], SUM X="6":
                        =>RESULT: No triplet found for the sum x=6 in the array: [2, 1, 5, 1, 6, 29, -6]
.
For ARRAY SOURCE: [12, 3, 4, 1, 6, 9, -6], SUM X="200":
                        =>RESULT: No triplet found for the sum x=200 in the array: [12, 3, 4, 1, 6, 9, -6]


For ARRAY SOURCE: [100, 5, 3, 1, 0, 5, -8], SUM X="24":
                        =>RESULT: No triplet found for the sum x=24 in the array: [100, 5, 3, 1, 0, 5, -8]


For ARRAY SOURCE: [12, 3, 0], SUM X="9":
                        =>RESULT: No triplet found for the sum x=9 in the array: [12, 3, 0]
.
For ARRAY SOURCE: [12, 3], SUM X="15":
                        =>RESULT: No triplet found for the sum x=15 in the array: [12, 3]


For ARRAY SOURCE: [12], SUM X="12":
                        =>RESULT: No triplet found for the sum x=12 in the array: [12]


For ARRAY SOURCE: [], SUM X="0":
                        =>RESULT: No triplet found for the sum x=0 in the array: []
.
For ARRAY SOURCE: [12, 3, 4, 1, 6, 9, -6], SUM X="24":
                        =>RESULT: [12,3 and 9] are TRIPLET for the given sum x=24 in the array: [12, 3, 4, 1, 6, 9, -6]


For ARRAY SOURCE: [100, -8, -1, -2, 0, -5, -8, -1, -4, 0, 2, -6, 4, 5, 0, -5, 2, -1, 0, -6, 7, 1, -200], SUM X="-192":
                        =>RESULT: [7,1 and -200] are TRIPLET for the given sum x=-192 in the array: [100, -8, -1, -2, 0, -5, -8, -1, -4, 0, 2, -6, 4, 5, 0, -5, 2, -1, 0, -6, 7, 1, -200]


For ARRAY SOURCE: [100, 3, 3, 1, 0, 5, -8, 9, 4, 3, 2, 6, 4, 5, 0, -5, 2, -1, 0, 6, 7, 1, -200], SUM X="106":
                        =>RESULT: [100,3 and 3] are TRIPLET for the given sum x=106 in the array: [100, 3, 3, 1, 0, 5, -8, 9, 4, 3, 2, 6, 4, 5, 0, -5, 2, -1, 0, 6, 7, 1, -200]


For ARRAY SOURCE: [100, 100, 3, 1, 0, 5, -8, 9, 4, 3, 2, 6, 4, 5, 0, -5, 2, -1, 0, 6, 7, 1, -200], SUM X="5":
                        =>RESULT: [3,1 and 1] are TRIPLET for the given sum x=5 in the array: [100, 100, 3, 1, 0, 5, -8, 9, 4, 3, 2, 6, 4, 5, 0, -5, 2, -1, 0, 6, 7, 1, -200]


For ARRAY SOURCE: [100, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 100], SUM X="15":
                        =>RESULT: [1,4 and 10] are TRIPLET for the given sum x=15 in the array: [100, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 100]


For ARRAY SOURCE: [12, 3, 4, 1, 6, 9, -6], SUM X="12":
                        =>RESULT: [12,6 and -6] are TRIPLET for the given sum x=12 in the array: [12, 3, 4, 1, 6, 9, -6]
.
For ARRAY SOURCE: [12, 3, 4], SUM X="19":
                        =>RESULT: [12,3 and 4] are TRIPLET for the given sum x=19 in the array: [12, 3, 4]
.
----------------------------------------------------------------------
Ran 5 tests in 0.003s

OK
```